What is Gradient Boosting

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. There are various ensemble methods such as stacking, blending, bagging and boosting. Gradient Boosting, as the name suggests is a boosting method.

Introduction

Boosting is loosely-defined as a strategy that combines multiple simple models into a single composite model. With the introduction of more simple models, the overall model becomes a stronger predictor. In boosting terminology, the simple models are called weak models or weak learners. Over the last years boosting techniques like AdaBoost and XGBoost have become much popular because of their great performance in online competitions like Kaggle. The two main boosting algorithms are Adaptive Boosting(AdaBoost) and Gradient Boosting.

What is Gradient Boosting?

The term gradient boosting consists of two sub-terms, gradient and boosting. We already know that gradient boosting is a boosting technique.Let us see how the term 'gradient' is related here.

Gradient boosting re-defines boosting as a numerical optimisation problem where the objective is to minimise the loss function of the model by adding weak learners using gradient descent. Gradient descent is a first-order iterative optimisation algorithm for finding a local minimum of a differentiable function. As gradient boosting is based on minimising a loss function, different types of loss functions can be used resulting in a flexible technique that can be applied to regression, multi-class classification, etc.

Intuitively, gradient boosting is a stage-wise additive model that generates learners during the learning process (i.e., trees are added one at a time, and existing trees in the model are not changed). The contribution of the weak learner to the ensemble is based on the gradient descent optimisation process. The calculated contribution of each tree is based on minimising the overall error of the strong learner.

Gradient boosting does not modify the sample distribution as weak learners train on the remaining residual errors of a strong learner (i.e, pseudo-residuals). By training on the residuals of the model, this is an alternative means to give more importance to misclassified observations. Intuitively, new weak learners are being added to concentrate on the areas where the existing learners are performing poorly. The contribution of each weak learner to the final prediction is based on a gradient optimisation process to minimise the overall error of the strong learner.

Difference between Gradient Boosting and Adaptive Boosting(AdaBoost)

Gradient boosting	Adaptive Boosting
This approach trains learners based upon minimising the loss function of a learner (i.e., training on the residuals of the model)	This method focuses on training upon misclassified observations. Alters the distribution of the training dataset to increase weights on sample observations that are difficult to classify.

Weak learners are decision trees constructed in a greedy manner with split points based on purity scores (i.e., Gini, minimise loss). Thus, larger trees can be used with around 4 to 8 levels. Learners should still remain weak and so they should be constrained (i.e., the maximum number of layers, nodes, splits, leaf nodes)	The weak learners incase of adaptive boosting are a very basic form of decision tree known as stumps.
All the learners have equal weights in the case of gradient boosting. The weight is usually set as the learning rate which is small in magnitude.	The final prediction is based on a majority vote of the weak learners' predictions weighted by their individual accuracy.

Understand Gradient Boosting Algorithm with example

Gradient Boosting is used for regression as well as classification tasks. In this section, we are going to see how it is used in regression with the help of an example. Following is a sample from a random dataset where we have to predict the weight of an individual, given the height, favourite colour, and gender of a person. Obviously favourite colour may seem irrelevant, but let us see if the learning algorithm figures it out too. In practice, it is better to remove such features but here we will go with the same. The target variable is shown in a red box while as features are shown in the green box.

Height(m)	Favourite Color	Gender	Weight(kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73

It builds a first learner to predict the observations in the training dataset. This learner is a basic learner. Usually for simplicity, we take an average of all the target variables and assume that to be predicted value in case of Regression as shown below.

Height(m)	Favourite Color	Gender	Weight(kg)	Prediction 1
1.6	Blue	Male	88	73.5
1.6	Green	Female	76	73.5
1.5	Blue	Female	56	73.5
1.8	Red	Male	73	73.5

Then, it calculates the loss (i.e., the value between the first learner's outcomes and the actual values).

Height(m)	Favourite Color	Gender	Weight(kg)	Prediction 1	Residuals(1)(New target variable)
1.6	Blue	Male	88	73.5	14.5
1.6	Green	Female	76	73.5	2.5
1.5	Blue	Female	56	73.5	-17.5
1.8	Red	Male	73	73.5	-0.5

It will build a second learner that is fitted/trained on the residual error usually known as pseudo-residual produced by the first learner to predict the loss after the first step and continue to do so until it reaches a threshold (i.e., residuals are zero).

Now calculate the new residuals using the new decision tree. Also, we use a learning rate of 0.1 to avoid big jumps.

New residual value for first sample =(88-73.5+0.1(14.5))=13.05. In a similar manner calculate the rest of the entries.

Height(m)	Favourite Color	Gender	Weight(kg)	Residuals(2)(New target variable)
1.6	Blue	Male	88	13.05
1.6	Green	Female	76	2.25

1.5	Blue	Female	56	-15.75
1.8	Red	Male	73	-0.45

Now using these residuals, we create a new tree and continue to this process till the loss is negligible. The next residuals are calculated by using the residuals of all the previous trees.

For example, the next residual of the first record is calculated as = (88- R) = 11.6

Where R = 73.5+0.1(14.5)) + 0.1(14.5))

By training the next learner on the gradient of the error with respect to the loss predictions of the previous learner, it is being trained to correct the mistakes of the previous model. You may notice that errors are gradually decreasing.

Note: In the above example, we have used a tree with 4 leaf nodes. In practice, we use trees with 8 to 34 leaves.

This is the core of gradient boosting and allows many simple learners to compensate for each other's weaknesses to better fit the data. There are some variants of gradient boosting and a few of them are briefly explained in the coming sections.

Extreme Gradient Boosting (XGBoost)

XGBoost is one of the most popular variants of gradient boosting. It is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. XGBoost is basically designed to enhance the performance and speed of a Machine Learning model. In prediction problems involving unstructured data (images, text, etc.), artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

XGBoost uses pre-sorted algorithm & histogram-based algorithm for computing the best split. The histogram-based algorithm splits all the data points for a feature into discrete bins and uses these bins to find the split value of the histogram. Also, in XGBoost, the trees can have a varying number of terminal nodes and left weights of the trees that are calculated with less evidence is shrunk more heavily.

LightGBM

LightGBM stands for lightweight gradient boosting machines. It uses a novel technique of Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value. LightGBM is prefixed as 'Light' because of its high speed. LightGBM is popular as it can handle the large size of data and takes lower memory to run.

Another reason why LightGBM is popular is as it focuses on the accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.

It can also handle categorical features by taking the input of feature names. It does not convert to one-hot coding and is much faster than one-hot coding. LGBM uses a special algorithm to find the split value of categorical features. Both LighGBM and XGBoost grow the trees leaf wise.

Cat Boost

Cat Boost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google's TensorFlow and Apple's Core ML. Cat Boost can work with diverse data types to help solve a wide range of problems that businesses face today.

Cat Boost has the flexibility of giving indices of categorical columns so that it can be encoded as one-hot encoding using one_hot_max_size (Use one-hot encoding for all features with a number of different values less than or equal to the given parameter value). Also If you don't pass any anything in cat_features argument, CatBoost will treat all the columns as numerical variables.

Catboost deals with categorical features by, "generating random permutations of the dataset and for each sample computing the average label value for the sample with the same category value placed before the given one in the

permutation". They also process the data with GPU acceleration and do feature discretisation into a fixed number of bins (128 and 32).

One main difference between CatBoost and other boosting algorithms is that the CatBoost implements symmetric trees. This may sound odd but it helps in decreasing prediction time, which is extremely important for low latency environments.

Gradient Boosting in Python

In this section, we are going to compare the performance of AdaBoost and Gradient boosting on a regression problem. Here we specifically use the diabetes dataset from the sk-learn library to compare the two algorithms. The dataset contains ten baseline variables, i.e., age, sex, body mass index, average blood pressure, and six blood serum measurements that were obtained for 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

In the following code, we use Python programming language and the sk-learn library for implementation.

```
from sklearn import datasets, ensemble
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
diabetes = datasets.load_diabetes()
X, y = diabetes.data, diabetes.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=13)

params = {'n_estimators': 500,
          'max_depth': 4,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
#gradient boosting classifier
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

#adaboost classifier
reg1=ensemble.AdaBoostRegressor()
reg1.fit(X_train, y_train)


mse = mean_squared_error(y_test, reg.predict(X_test))
print("The mean squared error (MSE) on test set for gradient boosting: {:.4f}".format(mse))

mse1 = mean_squared_error(y_test, reg1.predict(X_test))
print("The mean squared error (MSE) on test set for adaboost : {:.4f}".format(mse1))
```

Output:

```
The mean squared error (MSE) on test set for gradient boosting: 3013.4834
The mean squared error (MSE) on test set for adaboost : 3191.6401
```

From the above results, we conclude that for this particular dataset and hyperparameters, gradient boosting outperformed AdaBoost, though this might always be the case.

This brings us to the end of this article where we have learned about Gradient Boosting, a little about its variants and implementation of Gradient Boosting with SK-learn.

If you wish to learn more about Python and the concepts of Machine Learning, upskill with Great Learning's PG Program Artificial Intelligence and Machine Learning.
